OSProj4 Scheduling Algorithms

by 潘禧辰，518021910497

Menu

Abstract

- 要求使用c编写一个Scheduling Algorithms Simulator，模拟FCFS, SJF, Priority, RR, Priority_rr五种调度方案的执行顺序

  拓展要求：计算每种方案下的平均周转时间，平均等待时间和平均响应时间。

Environment

- Ubuntu 18.04
- Linux 5.3.0-42-generic
- VMware Workstation Rro 15.5.0 build-14665864

Quick Start

编译

osc10e的源代码中已经写好了Makefile文件，只需要相应执行指令即可：

```
make fcfs
make sjf
make priority
make rr
make priority_rr
```

测试代码

使用以下代码对5个调度算法进行测试

```
./fcfs schedule.txt
./sjf schedule.txt
./priority pri-schedule.txt
./rr rr-schedule.txt
./priority_rr schedule.txt
```

Implementation & Result

5个调度算法使用的头文件和add函数实现是相同的，新建一个`Task`，完成初始化后使用`insert`函数将其插入`tasks`列表即可：

```
#include "task.h"
#include "list.h"
#include "cpu.h"
#include "schedulers.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void add(char *name, int priority, int burst)
{
    Task *t = malloc(sizeof(Task));

    t->name = name;
    t->priority = priority;
    t->burst = burst;
    t->flag = 0;
    t->start = 0;
    t->end = 0;

    insert(&tasks, t);
}
```

这里因为涉及到计算平均周转时间，平均等待时间和平均响应时间，在rr和priority_rr算法中需要记录任务开始时间`start`和上次执行结束时间`end`以及任务是否首次执行的标志`flag`，所以对`task.h`文件中的`Task`定义做了一些修改，加入了这三个参量：

```
typedef struct task {
    char *name;
    int tid;
    int priority;
    int burst;
    int flag;
    int start;
    int end;
} Task;
```

下边分别介绍五种算法的`schedule`函数设计，因为需要输出平均周转时间，平均等待时间和平均响应时间，将`schedule`函数的声明做了修改：

```
Time* schedule();
```

新建了一个struct`Time`用于传递参数。

```
typedef struct Time
{
    float average_turnaround_time;
    float average_waiting_time;
    float average_response_time;
}Time;
```

同时对`drier.c`中的`main`函数也做了修改。首先是加入了`cnt`变量，每次插入task的时候递增，记录task的数量。用`time`接收`schedule`传回的`Time`并将其处理后输出：

```
int main(int argc, char *argv[])
{
    FILE *in;
    char *temp;
    char task[SIZE];

    char *name;
    int priority;
    int burst;
    int cnt = 0;

    in = fopen(argv[1],"r");

    while (fgets(task,SIZE,in) != NULL) {
        temp = strdup(task);
        name = strsep(&temp,",");
        priority = atoi(strsep(&temp,","));
        burst = atoi(strsep(&temp,","));

        // add the task to the scheduler's list of tasks
        add(name,priority,burst);
        cnt++;

        free(temp);
    }

    fclose(in);
```

```
    Time *time = (Time*)malloc(sizeof(Time));
    // invoke the scheduler
    time = schedule();

    printf("Average turnaround time: %f\n", time->average_turnaround_time /
cnt);
    printf("Average waiting time: %f\n", time->average_waiting_time / cnt);
    printf("Average response time: %f\n", time->average_response_time / cnt);

    return 0;
}
```

下面分别介绍五种算法的`schedule`函数设计

FCFS

整体的思路是初始化`Time`，之后按照FCFS的思想进行操作：

1. `head`指针指向链表尾最先来的任务
2. 将该任务的起始时间设定为目前时间，将之前这段时间计入总等待时间和总响应时间
3. 调用`run`执行
4. 更新目前时间`cu_time`
5. 将之前这段时间计入总周转时间
6. 完成执行，删除该任务
7. 读取下一个任务，回到第一步直到任务列表清空

```
struct node *tasks = NULL;

Time* schedule()
{
    Time *time = (Time*)malloc(sizeof(Time));
    time->average_turnaround_time = 0;
    time->average_waiting_time = 0;
    time->average_response_time = 0;
    int cu_time = 0;

    struct node *head = tasks;

    while(tasks != NULL){
        head = tasks;

        while(head->next != NULL)
            head = head -> next;

        head->task->start = cu_time;
        time->average_waiting_time += cu_time;
        time->average_response_time += cu_time;
        run(head->task, head->task->burst);

        cu_time += head->task->burst;
        time->average_turnaround_time += cu_time;
        delete(&tasks, head->task);
    }
    return time;
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/4$ ./fcfs schedule.txt
Running task = [T1] [4] [20] for 20 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T6] [1] [10] for 10 units.
Running task = [T7] [3] [30] for 30 units.
Running task = [T8] [10] [25] for 25 units.
Average turnaround time: 94.375000
Average waiting time: 73.125000
Average response time: 73.125000
pan@pan-virtual-machine:~/桌面/osproj/4$
```

SJF

整体的思路是初始化`Time`，之后按照SJF的思想进行操作：

1. 遍历链表，通过比较将`head`指针指向链表中burst time最短的任务
2. 将该任务的起始时间设定为目前时间，将之前这段时间计入总等待时间和总响应时间
3. 调用`run`执行
4. 更新目前时间`cu_time`
5. 将之前这段时间计入总周转时间
6. 完成执行，删除该任务
7. 读取下一个任务，回到第一步直到任务列表清空

```
struct node *tasks = NULL;

Time* schedule()
{
    Time *time = (Time*)malloc(sizeof(Time));
    time->average_turnaround_time = 0;
    time->average_waiting_time = 0;
    time->average_response_time = 0;
    int cu_time = 0;

    struct node *tmp, *head;

    while(tasks != NULL)
    {
        tmp = head = tasks;

        while(tmp != NULL) {
            if (tmp->task->burst < head->task->burst)
                head = tmp;
            tmp=tmp->next;
        }

        head->task->start = cu_time;
        time->average_waiting_time += cu_time;
        time->average_response_time += cu_time;
        run(head->task, head->task->burst);

        cu_time += head->task->burst;
        time->average_turnaround_time += cu_time;
        delete(&tasks, head->task);
    }
    return time;
```

```
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/4$ ./sjf schedule.txt
Running task = [T6] [1] [10] for 10 units.
Running task = [T4] [5] [15] for 15 units.
Running task = [T5] [5] [20] for 20 units.
Running task = [T1] [4] [20] for 20 units.
Running task = [T8] [10] [25] for 25 units.
Running task = [T3] [3] [25] for 25 units.
Running task = [T2] [3] [25] for 25 units.
Running task = [T7] [3] [30] for 30 units.
Average turnaround time: 82.500000
Average waiting time: 61.250000
Average response time: 61.250000
pan@pan-virtual-machine:~/桌面/osproj/4$
```

Priority

和SJF实现相同，只不过现在`head`指针指向的是优先级最高的任务

```
struct node *tasks = NULL;

Time* schedule()
{
    Time *time = (Time*)malloc(sizeof(Time));
    time->average_turnaround_time = 0;
    time->average_waiting_time = 0;
    time->average_response_time = 0;
    int cu_time = 0;

    struct node *tmp, *head;

    while(tasks != NULL){
        tmp = head = tasks;

        while(tmp != NULL) {
            if (tmp->task->priority > head->task->priority) {
                head = tmp;
            }
            tmp=tmp->next;
        }

        head->task->start = cu_time;
        time->average_waiting_time += cu_time;
        time->average_response_time += cu_time;
        run(head->task, head->task->burst);

        cu_time += head->task->burst;
        time->average_turnaround_time += cu_time;
        delete(&tasks, head->task);
    }
    return time;
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/4$ ./priority pri-schedule.txt
Running task = [T6] [1] [50] for 50 units.
Running task = [T5] [1] [50] for 50 units.
Running task = [T4] [1] [50] for 50 units.
Running task = [T3] [1] [50] for 50 units.
Running task = [T2] [1] [50] for 50 units.
Running task = [T1] [1] [50] for 50 units.
Average turnaround time: 175.000000
Average waiting time: 125.000000
Average response time: 125.000000
pan@pan-virtual-machine:~/桌面/osproj/4$
```

RR

整体的思路是初始化`Time`，之后按照RR的思想进行操作：

1. 将当前任务`cu_task`指针指向链表中的头结点
2. 更新`pos`，如果到达任务列表尾，那么回到头部重新开始
3. 判断burst time大小是否足够一个时间片，确定运行时间`run_time`
4. 调用`run`执行，如果是第一次执行那么将该任务的起始时间设定为目前时间，将之前这段时间计入总等待时间和总响应时间；如果不是第一次执行，那么将上次执行到目前为止的时间计入总等待时间
5. 更新目前时间`cu_time`和该任务burst time和上次执行结束时间`end`
6. 如果任务完成执行，那么将之前这段时间计入总周转时间并
7. 读取下一个任务，回到第一步直到任务列表清空

```
struct node *tasks = NULL;

Time* schedule()
{
    Time *time = (Time*)malloc(sizeof(Time));
    time->average_turnaround_time = 0;
    time->average_waiting_time = 0;
    time->average_response_time = 0;
    int cu_time = 0;

    struct node *cu_task, *pos = tasks;
    int run_time;

    while (tasks != NULL) {
        cu_task = pos;

        if (pos->next != NULL)
            pos = pos->next;
        else pos = tasks;

        if (QUANTUM < cu_task->task->burst)
            run_time = QUANTUM;
        else
            run_time = cu_task->task->burst;

        if (cu_task->task->flag == 0)
        {
            cu_task->task->flag = 1;
            cu_task->task->start = cu_time;
            time->average_waiting_time += cu_time;
            time->average_response_time += cu_time;
        }
        else
```

```
        {
            time->average_waiting_time += cu_time - cu_task->task->end;
        }
        run(cu_task->task, run_time);
        cu_time += run_time;
        cu_task->task->burst -= run_time;
        cu_task->task->end = cu_time;

        if (cu_task->task->burst == 0) {
            delete(&tasks, cu_task->task);
            time->average_turnaround_time += cu_task->task->end;
        }
    }
    return time;
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/4$ ./rr rr-schedule.txt
Running task = [T6] [40] [50] for 10 units.
Running task = [T5] [40] [50] for 10 units.
Running task = [T4] [40] [50] for 10 units.
Running task = [T3] [40] [50] for 10 units.
Running task = [T2] [40] [50] for 10 units.
Running task = [T1] [40] [50] for 10 units.
Running task = [T6] [40] [40] for 10 units.
Running task = [T5] [40] [40] for 10 units.
Running task = [T4] [40] [40] for 10 units.
Running task = [T3] [40] [40] for 10 units.
Running task = [T2] [40] [40] for 10 units.
Running task = [T1] [40] [40] for 10 units.
Running task = [T6] [40] [30] for 10 units.
Running task = [T5] [40] [30] for 10 units.
Running task = [T4] [40] [30] for 10 units.
Running task = [T3] [40] [30] for 10 units.
Running task = [T2] [40] [30] for 10 units.
Running task = [T1] [40] [30] for 10 units.
Running task = [T6] [40] [20] for 10 units.
Running task = [T5] [40] [20] for 10 units.
Running task = [T4] [40] [20] for 10 units.
Running task = [T3] [40] [20] for 10 units.
Running task = [T2] [40] [20] for 10 units.
Running task = [T1] [40] [20] for 10 units.
Running task = [T6] [40] [10] for 10 units.
Running task = [T5] [40] [10] for 10 units.
Running task = [T4] [40] [10] for 10 units.
Running task = [T3] [40] [10] for 10 units.
Running task = [T2] [40] [10] for 10 units.
Running task = [T1] [40] [10] for 10 units.
Average turnaround time: 275.000000
Average waiting time: 225.000000
Average response time: 25.000000
pan@pan-virtual-machine:~/桌面/osproj/4$
```

Priority_rr

和RR代码类似，只不过在Priority_rr中将任务列表`tasks`修改为指针列表，每个优先级对应单独一个任务列表。

```
struct node *tasks[MAX_PRIORITY+1];

Time* schedule()
{
    Time *time = (Time*)malloc(sizeof(Time));
    time->average_turnaround_time = 0;
    time->average_waiting_time = 0;
```

```
    time->average_response_time = 0;
    int cu_time = 0;
    struct node *cu_task, *pos;
    int run_time;

    for (int i = MAX_PRIORITY; i >= MIN_PRIORITY; i--)
    {
        if (tasks[i] == NULL)
            continue;

        pos = tasks[i];

        while (tasks[i] != NULL) {
            cu_task = pos;

            if (pos->next != NULL)
                pos = pos->next;
            else pos = tasks[i];

            if (QUANTUM < cu_task->task->burst)
                run_time = QUANTUM;
            else
                run_time = cu_task->task->burst;

            if (cu_task->task->flag == 0)
            {
                cu_task->task->flag = 1;
                cu_task->task->start = cu_time;
                time->average_waiting_time += cu_time;
                time->average_response_time += cu_time;
            }
            else
            {
                time->average_waiting_time += cu_time - cu_task->task->end;
            }
            run(cu_task->task, run_time);
            cu_time += run_time;
            cu_task->task->burst -= run_time;
            cu_task->task->end = cu_time;

            if (cu_task->task->burst == 0) {
                delete(&tasks[i], cu_task->task);
                time->average_turnaround_time += cu_task->task->end;
            }
        }
    }
    return time;
}
```

结果

```
pan@pan-virtual-machine:~/桌面/osproj/4$ ./priority_rr schedule.txt
Running task = [T8] [10] [25] for 10 units.
Running task = [T8] [10] [15] for 10 units.
Running task = [T8] [10] [5] for 5 units.
Running task = [T5] [5] [20] for 10 units.
Running task = [T4] [5] [15] for 10 units.
Running task = [T5] [5] [10] for 10 units.
Running task = [T4] [5] [5] for 5 units.
Running task = [T1] [4] [20] for 10 units.
Running task = [T1] [4] [10] for 10 units.
Running task = [T7] [3] [30] for 10 units.
Running task = [T3] [3] [25] for 10 units.
Running task = [T2] [3] [25] for 10 units.
Running task = [T7] [3] [20] for 10 units.
Running task = [T3] [3] [15] for 10 units.
Running task = [T2] [3] [15] for 10 units.
Running task = [T7] [3] [10] for 10 units.
Running task = [T3] [3] [5] for 5 units.
Running task = [T2] [3] [5] for 5 units.
Running task = [T6] [1] [10] for 10 units.
Average turnaround time: 106.875000
Average waiting time: 85.625000
Average response time: 68.750000
pan@pan-virtual-machine:~/桌面/osproj/4$
```

Difficulties

- 整体比较简单，就是对课本内容的实现。但因为这是第一个include了多个头文件的project，修改经常在多个文件中做出，没有工程经验一开始上手会有些手忙脚乱。

Reference

- Operating System Concept 10^{th} edition
- Source code for the 10th edition of Operating System Concepts https://github.com/greggagne/osc10e